On Safety in Discretionary Access Control

Ninghui Li Mahesh V. Tripunitara
Department of Computer Sciences and CERIAS
Purdue University
West Lafayette, IN, USA
{ninghui,tripunit}@cerias.purdue.edu

Abstract

An apparently prevailing myth is that safety is undecidable in Discretionary Access Control (DAC); therefore, one needs to invent new DAC schemes in which safety analysis is decidable. In this paper, we dispel this myth. We argue that DAC should not be equated with the Harrison-Ruzzo-Ullman access matrix scheme [10], in which safety is undecidable. We present an efficient (running time cubic in its input size) algorithm for deciding safety in the Graham-Denning DAC scheme [8], which subsumes the DAC schemes used in the literature on comparing DAC with other access control models. We also counter several claims made in recent work by Solworth and Sloan [27], in which the authors present a new access control scheme based on labels and relabelling and assert that it can implement the full range of DAC models. We present a precise characterization of their access control scheme and show that it does not adequately capture a relatively simple DAC scheme.

1. Introduction

Safety analysis, first formulated by Harrison, Ruzzo, and Ullman [10] for the access matrix model [13, 8], has been recognized as a fundamental problem in access control. Safety analysis decides whether rights can be leaked to unauthorized principals in future states. Safety analysis was shown to be undecidable in the HRU scheme. Since then, considerable research effort has gone into designing access control schemes in which safety analysis is decidable [1, 2, 5, 11, 17, 19, 20, 23, 24, 25, 26, 27, 29, 30]. Safety analysis is particularly interesting in DAC [6, 7, 8, 9], in which a subject gets rights to resources at the discretion of other subjects. Recently, there appears to be renewed interest in the topic of safety in DAC, as evidenced by the work by Solworth and Sloan [27], which was published at the IEEE Symposium on Security and Privacy in 2004. In that work, the authors assert that, in general, safety is undecidable in DAC, and use this assertion as the motivation for introducing a new access control scheme based on labels and relabelling that has decidable safety properties.

Our goals in this paper are to present a clear picture of safety in DAC and to counter several claims in Solworth and Sloan [27], that we demonstrate to be erroneous. The work in Solworth and Sloan [27] is based on the premise that safety is undecidable in DAC; therefore, one needs to design new schemes for DAC so that safety analysis is decidable. We assert that this premise is a myth, and conjecture that the basis for this myth is that DAC is sometimes erroneously equated to the HRU scheme [10] (for instance, in work such as [18, 22]). As we discuss in Section 3, DAC cannot be equated to the HRU scheme for the following reasons. First, the HRU scheme can be used to encode schemes that are not DAC schemes; therefore, the fact that safety is undecidable in the HRU scheme should not lead one to conclude that safety is undecidable in DAC. Second, features in DAC cannot always be encoded in the HRU scheme. For example, some DAC schemes require that each object be owned by exactly one subject; thus removal of a subject who has the ownership of some objects requires the transfer of ownership to some other subject (often times the owner of the subject being removed) so that this property is maintained. Both the removal of the subject and the transfer of ownership of objects it owns occur in a single state-change. A single HRU command cannot capture these features, because it cannot loop over all objects owned by a subject.

We dispel the myth that safety is undecidable in DAC by presenting an efficient algorithm for deciding safety in the DAC scheme proposed by Graham and Denning [8]. Our algorithm runs in time cubic in the size of the input. The Graham-Denning scheme is, to our knowledge, the first DAC scheme to have been proposed, and several other DAC schemes proposed subsequently are either subsumed by or are simple extensions of the Graham-Denning scheme. Examples of such DAC schemes include those used by Osborn et al. [21] to show that RBAC can be used to implement DAC. The same schemes are used by Solworth and Sloan [27] to show that the Solworth-Sloan scheme can implement DAC. Our algorithm suggests that safety in these DAC schemes can be efficiently decided and there is no need to invent new access control schemes with decidable safety as the primary goal.

Some may hold the view that safety can be trivially decided in DAC schemes. For instance, if the owner of an object is untrusted, then he can grant rights over the object to any other subject. Therefore, if such an owner exists, then the system will be unsafe for that object. While it may be easy to identify one or two such conditions that make a DAC system unsafe, identifying all such conditions may not be trivial. To our knowledge, algorithms for deciding safety in the Graham-Denning or other derived DAC schemes have not appeared in the literature. The proof that our algorithm is correct, which is in our technical report [15], was not trivial for us.

We observe that the presentation in [27] does not clearly specify what information is maintained in a state, how states may change, and the precise construction to implement DAC in the scheme. In this paper we give a precise characterization of the Solworth-Sloan scheme and an implementation of the SDCO scheme [21] in it. (Solworth and Sloan [27] use the word "implement" in this context, and therefore, we do the same. In previous work in the comparison of different access control schemes, "simulation" appears to be the preferred terminology.) We believe that a precise characterization of the Solworth-Sloan scheme is of interest independent of an assessment of its effectiveness in implementing other DAC schemes. The publication of two papers [27, 28] based on this scheme in recent major security conferences reflects that there is interest in such a access control scheme based on labels and relabelling.

Our precise characterization enables us to assess how effectively the Solworth-Sloan scheme implements the SDCO scheme. We counter several claims from Solworth and Sloan [27], and demonstrate that the claims are erroneous. Solworth and Sloan [27] claim that theirs is the first general access control model which both has a decidable safety property and is able to implement the full range of DAC models. We show that the proposed implementation of DAC schemes in the Solworth-Sloan scheme has significant deficiencies. Two particular limitations that we discuss are the lack of support for removing subjects and objects and the inability to ensure that an object has only one owner, as required by DAC schemes such as Strict DAC with Change of Ownership (SDCO), which is a simplified version of the Graham-Denning scheme. We observe also that the implementation incurs considerable overhead. Essentially for each new object to be created, a data structure of the size exponential in the total number of rights needs to be created.

The remainder of this paper is organized as follows. We discuss related work in Section 2 and give precise definitions of safety analysis in DAC in Section 3. In Section 4, we study safety analysis in the Graham-Denning scheme. We analyze the Solworth-Sloan scheme in Section 5 and conclude in Section 6.

2. Related Work

There is considerable work on DAC and safety analysis. To our knowledge, Graham and Denning [8] proposed the first DAC scheme. Their scheme is based on the work by Lampson on the access matrix model [13]. Subsequently, Griffiths and Wade proposed their DAC scheme for relational database systems [9]. Downs et al. [7] discussed salient aspects of DAC, and their work was subsequently subsumed by the NCSC's guide to DAC [6]. Lunt [18] examined various issues in DAC as part of broader work on issues in access control. Samarati and de Capitani di Vimercati [22] included discussions on DAC in their treatment of access control. Osborn et al. [21] discussed several DAC schemes that are sub-cases or variants of the Graham-Denning scheme in their comparison of DAC to RBAC. DAC was extended to include temporal constructs by Bertino et al. [3, 4]. Solworth and Sloan [27] presented a new DAC scheme based on labels and relabelling rules. The same scheme was also used by Solworth and Sloan in [28].

Safety is a fundamental property that was first proposed in the context of access control by Harrison et al. [10]. Subsequently, there has been considerable work on safety in various contexts related to security [1, 2, 5, 11, 14, 16, 17, 19, 20, 23, 24, 25, 26, 27, 29, 30]. Recent work by Li et al. [14, 16] perceived various forms of safety as special cases of more general security properties, and safety analysis is subsumed by security analysis. In this paper, we adopt this perspective in defining safety analysis in the next section. To our knowledge, the work by Solworth and Sloan [27] was the first to directly address safety in DAC. Other work on safety has been on specific schemes such as the HRU scheme [10], the ESPM scheme [1] and a trust management scheme [16]. Furthermore, to our knowledge, there is no prior work on safety analysis in the context of specific DAC schemes such as the Graham-Denning scheme [8].

3. Defining Safety Analysis in DAC

In this section, we define access control schemes and systems, and the general problem of security analysis in the context of such schemes and systems. We then define safety analysis as a special case of security analysis. In our definitions, we adopt the meta-formalism introduced by Li et al. [16, 14].

Definition 1 (Access Control Schemes and Systems) An access control scheme is a four-tuple $\langle \Gamma, \Psi, Q, \vdash \rangle$, where Γ is a set of states, Ψ is a set of state-change rules, Q is a set of queries and $\vdash : \Gamma \times Q \rightarrow \{true, false\}$ is the entailment function, that specifies whether a propositional logic formula of queries is true or not in a state.

A state-change rule, $\psi \in \Psi$, determines how the access control system changes state. Given two states γ and γ_1 and a state-change rule ψ, we write $\gamma \mapsto_\psi \gamma_1$ if the change from γ to γ_1 is allowed by ψ, and $\gamma \overset{*}{\mapsto}_\psi \gamma_1$ if a sequence of zero or more allowed state changes leads from γ to γ_1.

An access control system based on a scheme is a state-transition system specified by the four-tuple $\langle \gamma, \psi, Q, \vdash \rangle$, where $\gamma \in \Gamma$ is the start (or current) state, and $\psi \in \Psi$ specifies how states may change.

We recognize that our formalism for schemes and systems is fairly abstract. Nonetheless, we need such a formalism to be able to represent disparate access control schemes, such as those based on the access matrix, role-based access control and trust management approaches. When we specify a particular access control scheme, we specify each component precisely, using constructs that are well-understood.

An example of an access control scheme is the HRU scheme [10], in which the state consists of a finite set of subjects, a finite set of objects, and an access matrix with a row for each subject and a column for each object. Each cell in the access matrix is the set of the rights a subject has over the corresponding object. Examples of queries, $q_1, q_2 \in Q$ in the HRU scheme are "$q_1 = r \in M[s, o]$" and "$q_2 = r' \in M[s, o]$". The queries q_1 and q_2 ask whether the subject s has the right r and r' over the object o, respectively. Given a state, γ, and a state-change rule, ψ, in an HRU system, let S_γ be the set of subjects that exist in the state, γ, O_γ be the set of objects that exist, $M_\gamma[\,]$ be the access matrix, and R_ψ be the set of rights in the system. Then, $\gamma \vdash q_1 \wedge \neg q_2$ if and only if $s \in S_\gamma \wedge o \in O_\gamma \wedge r \in M_\gamma[s, o] \wedge r' \notin M_\gamma[s, o]$.

One of the components of our characterizations of security and safety analysis below warrants some explanation. Each instance of the analysis is associated with a set $\mathcal{T}$ of trusted subjects. The meaning of a trusted subject is that we preclude state-changes initiated by any subject from $\mathcal{T}$ in our analysis. The intuition is that we expect these subjects to be "well-behaved". That is, while such subjects may effect state-changes, they do so in such a way that the state that results from the state-changes they effect satisfies desirable properties (e.g., safety). Harrison et al. [10] do consider trusted subjects as part of their safety analysis. Nonetheless, as pointed out previously by Li et al. [16], the way they deal with trusted subjects is incorrect. They require that we delete the rows and columns corresponding to trusted subjects prior to the analysis. While a trusted subject is not allowed to initiate a state-change, she may be used as an intermediary, and the way Harrison et al. [10] deal with trusted subjects does not consider this possibility. In this paper, we require only that a member of the set of trusted subjects not initiate a state-change. In all other ways, these subjects continue to be part of the system.

Definition 2 (Security Analysis) Given an access control scheme $\langle \Gamma, \Psi, Q, \vdash \rangle$, a security analysis instance is of the form $\langle \gamma, \psi, \mathcal{T}, \Box\phi \rangle$, where ϕ is a propositional logic formula of queries and $\Box$ stands for "in the current and all future states," and is an operator from temporal logic [12]. Given such an instance, we say that the instance is true if for all states γ' such that $\gamma \overset{*}{\mapsto}_\psi \gamma'$, $\gamma' \vdash \phi$. That is, ϕ represents a security invariant that must be satisfied in all states reachable from γ under ψ, with no state change initiated by a user from the set $\mathcal{T}$, for the instance to be true. Otherwise, the instance is false.

Harrison et al. [10] informally characterize safety as the condition "that a particular system enables one to keep one's own objects 'under control' ". This informal characterization seems to be appropriate as a security property of interest in DAC systems, as the very purpose of DAC is that subjects should be able to keep objects that they own, under their control. More formally, safety analysis is a special case of security analysis, where the invariant is that an unauthorized subject should not have a particular right to a given object.

Definition 3 (Safety Analysis) Given an access control scheme $\langle \Gamma, \Psi, Q, \vdash \rangle$, let the set of subjects that can exist in a system based on the scheme be $\mathcal{S}$, let the set of objects be $\mathcal{O}$, and let the set of rights be $\mathcal{R}$. Assume that there exists a function $\mathsf{hasRight}$: $\mathcal{S} \times \mathcal{O} \times \mathcal{R} \rightarrow \{true, false\}$ such that $\mathsf{hasRight}(s, o, r)$ returns $true$ if in the current state, s and o exist, r is a right in the system, and s has the right r over o, and false otherwise. A safety analysis instance has the form $\langle \gamma, \psi, \mathcal{T}, \Box\neg\mathsf{hasRight}(s, o, r) \rangle$ for some $s \in \mathcal{S}$, $o \in \mathcal{O}$ and $r \in \mathcal{R}$. That is, safety analysis is security analysis with ϕ instantiated to $\neg\mathsf{hasRight}(s, o, r)$. The safety analysis instance is true if $\mathsf{hasRight}(s, o, r)$ is false in every reachable state, with no state change initiated by a user from $\mathcal{T}$, and false otherwise.

What is DAC? The NCSC guide titled 'A Guide To Understanding Discretionary Access Control in Trusted Systems' [6], portions of which were published as a research paper [7], states that "the basis for (DAC) is that an individual user, or program operating on the user's behalf, is allowed to specify explicitly the types of access other users (or programs executing on their behalf) may have to information under the user's control." We point out two specific properties from this characterization of DAC: (1) The notion of "control" – there is a notion that users exercise control over resources in that a user that controls a resource gets to dictate the sorts of rights other users have over the resource, and (2) the notion of initiation of an action by a user to change the protection state – such state changes occur because particular users initiate such changes. A representation of a DAC scheme needs to capture both these properties.

Some literature (for example, [18, 22]) appears to equate DAC with the HRU scheme [10]. This is incorrect, as there exist many systems based on the HRU scheme that are not DAC systems. For instance, consider an HRU system in which there is only one command, and that command has no condition. This system is not a DAC system as it does not have the first property from above on the control of resources by a subject. In addition, there are DAC schemes that do not have natural representations as HRU schemes. For instance, the Graham-Denning scheme [8] (see Section 4.1) is a DAC scheme in which a subject may be 'owned' or 'controlled' by at most one other subject. A system based on the HRU scheme cannot capture this feature in a natural way.

4. Safety Analysis in the Graham-Denning Scheme

In this section, we study safety analysis in the Graham-Denning DAC scheme [8]. We first present a description of the scheme in the following section. Our description clearly describes the states and state-change rules in the scheme. In Section 4.2, we present an algorithm to decide safety in the scheme, and show that the algorithm is correct. We also assert that the algorithm is efficient.

4.1. The Graham-Denning Scheme

In this section, We present a precise representation for the Graham-Denning scheme. We define what data are stored in a protection state, and how a state-change rule changes a state.

Assumptions We postulate the existence of the following countably infinite sets: $\mathcal{O}$, the set of objects; $\mathcal{S}$, the set of subjects ($\mathcal{S} \subset \mathcal{O}$); and $\mathcal{R}$, the set of rights.

Note that the set of objects (or subjects) in any given state in the Graham-Denning scheme is finite; however, the number of objects that could be added in some future state is unbounded. Similarly, the set of rights in any given access control system is finite; however, different access control systems may use different sets of rights. Therefore, we assume $\mathcal{S}$, $\mathcal{O}$, and $\mathcal{R}$ are countably infinite.

We assume a naming convention so that we can determine, in constant time, whether a given object, o, is a subject (i.e., $o \in \mathcal{S}$) or not (i.e., $o \in \mathcal{O} - \mathcal{S}$). There exists a special "universal subject" $\mathsf{u} \in \mathcal{S}$; the role of u will be explained later. The set of rights $\mathcal{R}$ contains two special rights, own and $control$, a countably infinite set $\mathcal{R}_b$ of "basic" rights, and a countably infinite set $\mathcal{R}_b^*$ of basic rights with the copy flag denoted by $*$, i.e., $\mathcal{R}_b^* = \{r^* | r \in \mathcal{R}_b\}$. In other words, $\mathcal{R} = \{own, control\} \cup \mathcal{R}_b \cup \mathcal{R}_b^*$. The meaning of the copy flag is clarified when we discuss the state-change rules for the scheme. An access control system based on the Graham-Denning scheme is associated with a protection state, and a state-change rule.

States, Γ A state in the Graham-Denning scheme, γ, is associated with the tuple $\langle O_\gamma, S_\gamma, M_\gamma[\,]\rangle$, where $O_\gamma \subset \mathcal{O}$ is a finite set of objects that exist in the state γ, $S_\gamma \subset \mathcal{S}$ is a finite set of subjects that exist in γ, and S_γ is a subset of O_γ. $M_\gamma[\,]$ is the access matrix, and $M_\gamma[\,]$: $S_\gamma \times O_\gamma \to 2^{\mathcal{R}}$. That is, $M_\gamma[s, o] \subset \mathcal{R}$ is the finite set of rights the subject $s \in S_\gamma$ has over the object $o \in O_\gamma$.

Every state, $\gamma = \langle O_\gamma, S_\gamma, M_\gamma[\ ]\rangle$, in the Graham-Denning scheme satisfies the following seven properties.

1. Every object must be owned by at least one subject, i.e., $\forall\, o \in O_\gamma\, \exists s \in S_\gamma(own \in M_\gamma[s, o])$.
2. Objects are not controlled, only subjects are, i.e., $\forall\, o \in (O_\gamma - S_\gamma)\, \forall\, s \in S_\gamma(control \notin M_\gamma[s, o])$.
3. The special subject u exists in the state, is not owned by any subject, and is not controlled by any other subject, i.e., $\mathsf{u} \in S_\gamma \wedge \forall\, s \in S_\gamma(own \notin M_\gamma[s, \mathsf{u}]) \wedge \forall\, s \in S_\gamma - \{\mathsf{u}\}(control \notin M_\gamma[s, \mathsf{u}])$.
4. A subject other than u is owned by exactly one other subject, i.e., for every $s \in S_\gamma - \{\mathsf{u}\}$, there exists exactly one $s' \in S_\gamma$ such that $s' \neq s$ and $own \in M_\gamma[s', s]$;
5. Every subject controls itself, i.e., $\forall s \in S_\gamma(control \in M_\gamma[s, s])$.
6. A subject other than u is controlled by at most one other subject, i.e., for every $s \in S_\gamma - \{\mathsf{u}\}$, there exists at most one $s' \in S_\gamma$ such that $s' \neq s$ and $control \in M_\gamma[s', s]$.
7. There exists no set of subjects such that they form a "cycle" in terms of ownership of each other (and in particular, a subject does not own itself), i.e., $\neg(\exists\, \{s_1, \ldots, s_n\} \subseteq S_\gamma(own \in M_\gamma[s_2, s_1] \wedge own \in M_\gamma[s_3, s_2] \wedge \cdots \wedge own \in M_\gamma[s_n, s_{n-1}] \wedge own \in M_\gamma[s_1, s_n]))$.

These state invariants are maintained by the state-change rules.

State-Change Rules, Ψ Each member, ψ, of the set of state-change rules, Ψ, in the Graham-Denning scheme, is a set of commands parameterized by a set of rights, R_ψ. These commands are shown in Figure 1. Where possible, we use the syntax for commands from the HRU scheme [10], but as we mention in Section 3, we cannot represent all aspects of DAC schemes using only constructs for commands in the HRU scheme. We use some additional well-known constructs such as $\forall$ and $\exists$ in these commands. A state-change is the successful execution of one of the commands. We assume that the state subsequent to the execution of a command is γ'. We denote such a state-change as $\gamma \mapsto_{\psi(s)} \gamma'$, where s is the initiator of the command. We point out that for each command, unless specified otherwise, $S_{\gamma'} = S_\gamma$, $O_{\gamma'} = O_\gamma$, and $M_{\gamma'}[s, o] = M_\gamma[s, o]$ for every $s \in S_\gamma$ and $o \in O_\gamma$. We use $\leftarrow$ to denote assignment, i.e., $x \leftarrow y$ means that the value in x is replaced with the value in y. The commands in the Graham-Denning scheme are the following. The first parameter to each command is named i, and is the subject that is the initiator of the execution of the command.

- transfer_r(i, s, o) This command is used to grant the right r by an initiator that has the right r^* over o. There is one such command for every $r \in R_\psi \cap \mathcal{R}_b$. The initiator, i, must possess the right r^* over o, and the subject s must exist for this command execution to succeed.
- transfer_r$^*(i, s, o)$ This command is used to grant the right r^* by an initiator that has the right r^* over o. There is one such command for every $r^* \in R_\psi \cap \mathcal{R}_b^*$. The initiator, i, must possess the right r^* over o, and the subject s must exist for this command execution to succeed.
- transfer_own(i, s, o) This command is used to transfer ownership over o from i to s. For this command to succeed, i must have the own right over o, s must exist, and the transfer of ownership must not violate invariant (7) from the list of state invariants we discuss above. After the execution of this command, i will no longer have the own right over o (but s will).
- grant_r(i, s, o) This command is used to grant the right r over o by the owner of o. There is one such command for every $r \in R_\psi \cap \mathcal{R}_b$. For this command execution to succeed, i must have the own right over o, and s must exist.
- grant_r$^*(i, s, o)$ This command is very similar to the previous command, except the the owner grants $r^* \in R_\psi \cap \mathcal{R}_b^*$.
- grant_control(i, s, o) This command is used to grant the $control$ right over o by its owner. For the execution of this command to succeed, i must have the right $control$ over o, s must exist, o must be a subject, and another subject must not already have the right $control$ over o. These checks are needed to maintain the state invariants related to the $control$ right that we discuss above.
- grant_own(i, s, o) This command is used to grant the own right over o. This is different from the transfer_own command in that in this case, i retains (joint) ownership over o. For the execution of this command to succeed, i must have the right own over o, o must not be a subject, and s must exist.
- delete_r(i, s, o) This command is used to delete a right a subject has over o. There is one such command for every $r \in R_\psi \cap \mathcal{R}_b$. For the execution of this command to succeed, i must have the right own over o, and s must exist.
- delete_r$^*(i, s, o)$ This command is similar to the previous command, except that a right $r^* \in R_\psi \cap \mathcal{R}_b^*$ is deleted.
- create_object(i, o) This command is used to create an object that is not a subject. For the execution of this command to succeed, i must exist, and o must be an object that is not a subject, that does not exist. An effect of this command is that i gets the own right over o in the new state.
- destroy_object(i, o) This command is used to destroy an object that exists. For the execution of this command to succeed, i must have the right own over o, and o must be an object that is not a subject.
- create_subject(i, s) This command is used to create a subject. For the execution of this command to succeed, i must exist, and s must be a subject that does not exist. In the new state, i has the own right over s, and s has the $control$ right over itself.
- destroy_subject(i, s) This command is used to destroy a subject. For the execution of this command to succeed, i must have the own right over s. An effect of this command is that ownership over any object owned by s is transferred to i.

4.2. Safety analysis

An algorithm to decide whether a system based on the Graham-Denning scheme is safe is shown in Figure 2. A system based on the Graham-Denning scheme is characterized by a start-state, γ, and state-change rule, ψ (which is a set of commands). The algorithm takes as input γ, ψ, a triple, $\omega = \langle s, o, x \rangle \in \mathcal{S} \times \mathcal{O} \times \mathcal{R}$, and a finite set, $\mathcal{T} \subset \mathcal{S}$, of trusted subjects. The algorithm outputs "true" if the system satisfies the safety property with respect to the subject s, object o and right x, and "false" otherwise. We first discuss the algorithm, and then its correctness and time-complexity.

In lines 5-10 of the algorithm, we check the cases for which we do not have to consider potential state-changes

$command$ $\mathsf{transfer\_r}(i, s, o)$
$if\ r^* \in M_\gamma[i, o] \wedge s \in S_\gamma\ then$
$M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] \cup \{r\}$

$command$ $\mathsf{transfer\_own}(i, s, o)$
$if\ own \in M_\gamma[i, o] \wedge o \in S_\gamma \wedge s \in S_\gamma\ then$
$if\ \nexists \{s_1, \ldots, s_n\} \in S_\gamma\ such\ that$
$own \in M_\gamma[s_1, s] \wedge own \in M_\gamma[s_2, s_1]$
$\wedge \cdots \wedge own \in M_\gamma[s_n, s_{n-1}]$
$\wedge\ own \in M_\gamma[o, s_n]\ then$
$M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] \cup \{own\}$
$M_{\gamma'}[i, o] \leftarrow M_\gamma[i, o] - \{own\}$

$command$ $\mathsf{grant\_control}(i, s, o)$
$if\ own \in M_\gamma[i, o] \wedge o \in S_\gamma \wedge s \in S_\gamma\ then$
$if \nexists\ s' \in S_\gamma\ such\ that$
$s' \neq o \wedge control \in M_\gamma[s', o]\ then$
$M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] \cup \{control\}$

$command$ $\mathsf{delete\_r}(i, s, o)$
$if\ \ (own \in M_\gamma[i, o] \wedge s \in S_\gamma)$
$\vee\ control \in M_\gamma[i, s]\ then$
$M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] - \{r\}$

$command$ $\mathsf{create\_object}(i, o)$
$if\ o \notin O_\gamma \wedge i \in S_\gamma \wedge o \in \mathcal{O} - \mathcal{S}\ then$
$O_{\gamma'} \leftarrow O_\gamma \cup \{o\}$
$M_{\gamma'}[i, o] \leftarrow own$

$command$ $\mathsf{create\_subject}(i, s)$
$if\ s \notin O_\gamma \wedge i \in S_\gamma \wedge s \in \mathcal{S}\ then$
$O_{\gamma'} \leftarrow O_\gamma \cup \{s\}$
$S_{\gamma'} \leftarrow S_\gamma \cup \{s\}$
$M_{\gamma'}[i, s] \leftarrow \{own\}$
$M_{\gamma'}[s, s] \leftarrow \{control\}$

$command$ $\mathsf{transfer\_r}^*(i, s, o)$
$if\ r^* \in M_\gamma[i, o] \wedge s \in S_\gamma\ then$
$M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] \cup \{r^*\}$

$command$ $\mathsf{grant\_r}(i, s, o)$
$if\ own \in M_\gamma[i, o] \wedge s \in S_\gamma\ then$
$M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] \cup \{r\}$

$command$ $\mathsf{grant\_r}^*(i, s, o)$
$if\, own \in M_\gamma[i, o] \wedge s \in S_\gamma\ then$
$M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] \cup \{r^*\}$

$command$ $\mathsf{grant\_own}(i, s, o)$
$if\ own \in M_\gamma[i, o] \wedge o \notin S_\gamma$
$\wedge\ s \in S_\gamma\ then$
$M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] \cup \{own\}$

$command$ $\mathsf{delete\_r}^*(i, s, o)$
$if\ \ (own \in M_\gamma[i, o] \wedge s \in S_\gamma)$
$\vee\ control \in M_\gamma[i, s]\ then$
$M_{\gamma'}[s, o] \leftarrow M_\gamma[s, o] - \{r^*\}$

$command$ $\mathsf{destroy\_object}(i, o)$
$if\ own \in M_\gamma[i, o] \wedge o \notin S_\gamma\ then$
$O_{\gamma'} \leftarrow O_\gamma - \{o\}$

$command$ $\mathsf{destroy\_subject}(i, s)$
$if\ own \in M_\gamma[i, s] \wedge s \in S_\gamma\ then$
$\forall\ o \in O_\gamma, if\ own \in M_\gamma[s, o]\ then$
$M_{\gamma'}[i, o] \leftarrow M_\gamma[i, o] \cup \{own\}$
$O_{\gamma'} \leftarrow O_\gamma - \{s\}$
$S_{\gamma'} \leftarrow S_\gamma - \{s\}$

Figure 1. The set of commands that constitutes the state-change rule, ψ, for a system based on the Graham-Denning scheme. Each command has a name (e.g., transfer_own**), and a sequence of parameters. The first parameter is always named i, and is the initiator of the command, i.e., the subject that executes the command. There is one** transfer_r, grant_r, **and** delete_r **command for each $r \in R_\psi \cap \mathcal{R}_b$, and one** transfer_r*, grant_r*, **and** delete_r* **command for each $r^* \in R_\psi \cap \mathcal{R}_b^*$.**

1 Subroutine isSafeGD $(\gamma, \psi, \omega, \mathcal{T})$
2 /* inputs: γ, ψ, $\omega = \langle s, o, x \rangle$, $\mathcal{T} \subseteq \mathcal{S}$ */
3 /* output: true or false */
4 if $x \in \mathcal{R}_b^*$ then let $y \leftarrow x$
5 else if $x \neq \mathit{own} \wedge x \neq \mathit{control}$ then let $y \leftarrow x^*$
6 else let $y \leftarrow \mathit{invalid}$ /* No copy flags for own or $\mathit{control}$ */
7 if $x \notin R_\psi$ then return true
8 if $x = \mathit{control} \wedge o \in \mathcal{O} - \mathcal{S}$ then return true
9 if $x \in M_\gamma[s, o]$ then return false
10 if $y \in M_\gamma[s, o]$ then return false
11 if $\mathcal{T} \supseteq S_\gamma$ then return true
12 if $o \notin O_\gamma$ then return false
13 if $\exists \widehat{s} \in S_\gamma - \mathcal{T}$ such that $y \in M_\gamma[\widehat{s}, o]$ then return false
14 for each sequence $\mathcal{U}, s_n, \ldots, s_2, s_1$ such that
15 $\mathit{own} \in M_\gamma[s_1, o] \wedge \cdots \wedge \mathit{own} \in M_\gamma[s_n, s_{n-1}] \wedge \mathit{own} \in M_\gamma[u, s_n]$ do
16 if $\exists s_i \in \{s_1, \ldots, s_n\}$ such that $s_i \in S_\gamma - \mathcal{T}$ then return false
17 return true

Figure 2. The subroutine isSafeGD returns "true" if the system based on the Graham-Denning scheme, characterized by the start-state, γ, and state-change rule, ψ, satisfies the safety property with respect to ω and $\mathcal{T}$. Otherwise, it returns "false". In line 6, we assign some invalid value to y, as there is not corresponding right with the copy flag for the rights own and $control$. In this case, the algorithm will not return in line 10 or 13. The subject u appears in line 15 only to emphasize that the "chain" of ownership is terminal.

before we are able to decide whether the system is safe or not. In lines 5-6, we consider the case that a subject may have (or acquire) the right with the copy flag. For this, we need to exclude own and $control$ from consideration, as those rights do not have counterparts with the copy flag. We use the mnemonic $invalid$ to indicate this. In line 7, we check that the right x is indeed in the system. In line 8, we check whether we are being asked whether s can get the $control$ right over o, where o is an object that is not a subject (we know s does not have and cannot get the right, by property (2) of the seven properties we discuss in the previous section). In line 9, we check whether the right x has already been acquired by s over o. In line 10, we check that if the right y has already been acquired by s over o (the check in line 10 is needed when $x \in \mathcal{R}_b$, as then, the possession of x^* implies the possession of x; in the case that $x \in \mathcal{R}_b^*$, the lines 9 and 10 are identical). When $x = own$ or $x = control$, the condition of line 10 will never be true, and we will not return from that line. In the remainder of the algorithm, we consider those cases in which a state-change is needed before s can get x over o (if it can at all). In line 11, we check whether there is at least one subject that can initiate state-changes, and if not, we know that the system is safe. In line 12, we check whether o exists, and if it does not, given that there exists a subject that can create o (from our check in line 11), the subject can then grant x to s over o. In line 13, we check whether there is a subject that can initiate state-changes, and that has x with the copy-flag (or x itself, if $x \in \mathcal{R}_b^*$). If $x = own$ or $x = control$, the condition of line 13 cannot be true. In lines 14-16, we check whether there is a sequence of subjects with the particular property that each owns the next in the sequence, and the last subject in the sequence owns o. If any one of those subjects can initiate state-changes, then we conclude that the system is not safe and return false. In all other cases, we conclude that the system is safe, and return true.

The following lemma asserts that the algorithm is correct. Theorem 2 summarizes our results with respect to safety analysis in the Graham-Denning scheme.

Lemma 1 *A system based on the Graham-Denning scheme, that is characterized by the start-state, γ, and state-change rule, ψ, is safe with respect to $\omega = \langle s, o, x \rangle$ and $\mathcal{T} \subset \mathcal{S}$ (where $\mathcal{T}$ is finite) if and only if* isSafeGD$(\gamma, \psi, \omega, \mathcal{T})$ *returns true.*

Proof. Sketch: the proof is quite lengthy, and we present it in [15]. We present a sketch of the proof here. For the "if" part, we need to show that if the system is not safe with respect to ω and $\mathcal{T}$, then isSafeGD returns false on input $(\gamma, \psi, \omega, \mathcal{T})$. If the system is not safe, then we know that there exists a state-change sequence $\gamma \mapsto_{\psi(s_1)} \gamma_1 \mapsto_{\psi(s_2)} \cdots \mapsto_{\psi(s_n)} \gamma_n$, such that $x \in M_{\gamma_n}[s, o]$. If such a sequence exists with $n = 0$, then this can only be because s already has the right, and we show that in this case the algorithm returns false. If $n = 1$, then the right has to appear in $M_{\gamma_1}[s,o]$ in only one state-change, and we show that in this case as well, the algorithm returns false. For the general case, we use induction on n, with $n = 1$ as the base case.

For the "only if" part, we need to show that if the algorithm returns false, then the system is not safe with respect to ω and $\mathcal{T}$. We consider each case in which the algorithm returns false (lines 9, 10, 12, 13 and 16). In each case, we construct a state-change sequence such that in the final state of the sequence, γ', $x \in M_{\gamma'}[s,o]$. ∎

Theorem 2 *Safety is efficiently decidable in a system based on the Graham-Denning scheme. In particular,* isSafeGD *runs in time at worst cubic in the size of the components of the start state and the set of rights in the system.*

Proof. We make the following observations about the running time of isSafeGD in terms of its input, namely, $S_\gamma, O_\gamma, R_\psi$, $M_\gamma[\,]$, ω and $\mathcal{T}$, by considering each line in the algorithm as follows. Each of the lines 5-10 runs in time at worst linear in the size of the input. In particular, as we mention in the previous section, we adopt a naming convention for subjects and objects that enables us to perform the check $o \in \mathcal{O} - \mathcal{S}$ in line 8, in constant time. Line 11 runs in time at worst quadratic in the size of the input ($|S_\gamma| \times |\mathcal{T}|$), line 12 runs in time at worst linear ($|O_\gamma|$), and line 13 runs in time at worst quadratic ($|S_\gamma| \times |R_\psi|$). As each subject is owned only by one other subject, each sequence to which line 14 refers is of size at most $|S_\gamma|$. Furthermore, there are at most $|S_\gamma|$ such sequences. Therefore, lines 14-16 run in time at worst cubic in the size of the input. The fact that isSafeGD$(\gamma, \psi, \omega, \mathcal{T})$ runs in time polynomial in the size of the input in conjunction with Lemma 1 proves our assertion. ∎.

We observe that cubic running time is only an upper-bound, and is not necessarily a tight upper-bound on the time-complexity of the algorithm. It may be possible, for instance, to store the "chains" of owners in some auxiliary data structure to get a faster running time.

5. The Solworth-Sloan Scheme, Revisited

Solworth and Sloan [27] present a new DAC scheme based on labels and relabelling rules, and we call it the Solworth-Sloan scheme. While the presentation in [27] does not clearly specify what information is maintained in a state and how states may change, we were able to infer what is intended after considerable effort.

In this section, we give a precise characterization of the Solworth-Sloan scheme as a state transition system. Our objective in doing so is to represent the Solworth-Sloan scheme sufficiently precisely to enable comparisons to other DAC schemes. In particular, our intent is to assess the mapping of DAC schemes to the Solworth-Sloan scheme that is discussed by Solworth and Sloan [27]. Solworth and Sloan [27] refer to the DAC schemes discussed by Osborn et al. [21] and assert that they present a general access control model which is sufficiently expressive to implement each of these DAC models. In this section, we show that this claim is incorrect.

We reiterate that the DAC schemes discussed by Osborn et al. [21] are either subsumed by, or are minor extensions of the Graham-Denning scheme that we discuss in Section 4. We have shown in Section 4.2 that safety is efficiently decidable in the Graham-Denning scheme, and our algorithm can be used with relatively minor modifications to decide safety in these schemes. Thereby, Solworth and Sloan's [27] other assertion in reference to the DAC schemes discussed by Osborn et al. [21], that "…every published general access control model…either is insufficiently expressive to represent the full range of DACs or has an undecidable safety problem…", has been rendered invalid.

5.1. The Solworth-Sloan Scheme

Overview There exists the following countably infinite sets of constants:

- a set $\mathcal{S}$ of subjects
- a set $\mathcal{O}$ of objects
- a set $\mathcal{R}$ of rights
- a set $\mathcal{G}$ of groups
- a set $\mathcal{T}^o$ of object tags
- a set $\mathcal{T}^g$ of group tags

An *object label* is a pair $\langle s, t\rangle$, where $s \in \mathcal{S}$ is a subject and $t \in \mathcal{T}^o$ is a object tag.

Which rights a subject has over a particular object are determined indirectly in the following three steps.

1. There is a labelling function label that assigns an object label to each object.

   An object's label may be changed by object relabelling rules, which determine whether an action rewriting one object label into another succeeds or not. For example, when the object label $\ell_1 = \langle s_1, t_1\rangle$ is relabelled to $\ell_2 = \langle s_2, t_2\rangle$, all objects that originally have the label ℓ_1 now have the label ℓ_2.

2. There is an authorization function auth that maps each object label and each right to a group. For each object label ℓ and each right r, members of the group identified by auth(ℓ, r) have right r over objects that are assigned the label ℓ.

3. Which subjects are members of a group is determined by native group sets (NGS's), which are complicated structures that we describe below. We define a function *members* that maps each group to a set of subjects.

We schematically illustrate the steps to determine whether a subject can access an object or not as follows.

$$\text{objects} \xrightarrow{\mathsf{label}} \text{object labels} \xrightarrow{\mathsf{auth}} \text{groups} \xrightarrow{members} \text{subjects}$$

States, Γ A state, γ, is characterized by a 9-tuple $\langle S_\gamma, O_\gamma, R_\gamma, G_\gamma, L_\gamma, \mathsf{label}_\gamma, \mathsf{auth}_\gamma, \mathsf{ORS}_\gamma, E_\gamma \rangle$.

- S_γ is the set of subjects in the state γ; O_γ is the set of objects in the state γ; R_γ is the set of rights in the state γ, and G_γ is the set of groups in state γ.

  There is a distinguished right wr, which exists in every state, i.e., $\mathsf{wr} \in R_\gamma$. The role of wr is explained in our discussion of the state-change rules.

- $L_\gamma \subset S_\gamma \times \mathcal{T}^o$ is the finite set of object labels in the state γ.

- label_γ: $O_\gamma \longrightarrow L_\gamma$ assigns a unique object label to each object in the current state.

- auth_γ : $(L_\gamma \times \mathcal{R}_\gamma) \longrightarrow G_\gamma$ maps each pair of an object label and a right to a group. For example, $\mathsf{auth}_\gamma[\ell, \mathsf{re}] = g_1$ means that the group g_1 has the re right over all objects labelled ℓ.

- ORS_γ is an ordered sequence of object relabelling rules, each rule has the form of $\mathsf{rl}(p_1, p_2) = h$, where rl is a keyword, and p_1, p_2 are object patterns. An *object pattern* is a pair, where the first element is a subject in $\mathcal{S}$ or one of the three special symbols $*$, $*\mathsf{u}$, and $*\mathsf{w}$, and the second element is an object tag in $\mathcal{T}^o$ or the special symbol $*$. In the rule $\mathsf{rl}(p_1, p_2) = h$, h is a group, a subject, or one of the four following sets: $\{\}, \{*\}, \{*\mathsf{u}\}, \{*\mathsf{w}\}$. When h is $\{*\mathsf{u}\}$ (resp., $\{*\mathsf{w}\}$), $\{*\mathsf{u}\}$ (resp., $\{*\mathsf{w}\}$) must appear in p_1 or p_2.

  For example, the following is an ORS_γ, in which s_1 is a subject, t_1 is an object tag, and g_1 is a group:

$$\begin{array}{rcl} \mathsf{rl}(\langle *\mathsf{u}, t_1\rangle,\ \langle s_1, *\rangle) & = & g_1 \\ \mathsf{rl}(\langle s_1, *\rangle,\ \langle *\mathsf{u}, t_2\rangle) & = & \{*\} \\ \mathsf{rl}(\langle *\mathsf{u}, *\rangle,\ \langle *\mathsf{u}, *\rangle) & = & \{*\mathsf{u}\} \\ \mathsf{rl}(\langle *\mathsf{u}, *\rangle,\ \langle *\mathsf{w}, *\rangle) & = & \{\} \end{array}$$

- E_γ is a finite set of native group sets (NGS's) that exist in the state, γ. Each $e \in E_\gamma$ is characterized by the 7-tuple $\langle e.G,\ e.T^g,\ e.\mathsf{gtag},\ e.nt^g,\ e.\mathsf{admin},\ e.\mathsf{patterns},\ e.\mathsf{GRS}\rangle$.

  – $e.G \subseteq G_\gamma$ is the set of groups that are defined in this NGS.

  – $e.T^g \subseteq \mathcal{T}^g$ is the set of group tags that are used in this NGS.

  – The function $e.\mathsf{gtag} : S_\gamma \longrightarrow e.T^g$ assigns a unique tag to each subject in the current state.

  – $e.nt^g$ is a group tag in $e.T^g$; it determines when a new subject is added to the state, which tag is assigned to that subject. That is, if a subject s is added, then $e.\mathsf{gtag}[s]$ would be set to $e.nt^g$.

  – $e.\mathsf{admin}$ points to one NGS in E_γ; it identifies an NGS in the current state as the administrative group set of the NGS e; $e.\mathsf{admin}$ could be e, in which case e is the administrative group set for itself.

  – $e.\mathsf{patterns}$ is a function mapping each group in $e.G$ to a (possibly empty) set of group patterns. Each *group pattern* is a pair where the first element is either a subject in the current state or a special symbol $*\mathsf{u}$, and the second element is a group tag in $e.T^g$. In other words, the set of all group patterns that can be used in e, denoted by $e.P^g$, is $(S_\gamma \cup \{*\mathsf{u}\}) \times e.T^g$, and the signature of $e.\mathsf{patterns}$ is $e.G \longrightarrow 2^{e.P^g}$, where $2^{e.P^g}$ denote the powerset of $e.P^g$.

    For any group $g \in e.G$, $e.\mathsf{patterns}[g]$ gives a set of patterns for determining memberships of the group. Intuitively, the label $\langle *\mathsf{u}, t^g\rangle$ is in $e.\mathsf{patterns}[g]$ means that any subject who is assigned (via the e.gtag function) the group tag t^g is a member of the group; and the label $\langle s, t^g\rangle$ is in $e.\mathsf{patterns}[g]$ means that the subject s is a member of the group if it is assigned the group tag t^g.

  – $e.\mathsf{GRS}$ is a set of group relabelling rules, each has the form $Relabel(t_1^g, t_2^g) = g$, where $Relabel$ is a keyword, $t_1^g, t_2^g \in e.T^g$ are two group tags used in this NGS, and g is a group defined in the administrative group set $e.\mathsf{admin}$ (i.e., $g \in e.\mathsf{admin}.G$). The role of a member of $e.\mathsf{GRS}$ is explained in the following discussion of state-change rules in the context of group_tag_relabel.

An additional constraint on the state γ is that each group is defined in exactly one NGS and each group tag can be used in at most one NGS, i.e.,

$$\forall e_1 \in E_\gamma \forall e_2 \in E_\gamma\ (\ e_1.G \cap e_2.g = \emptyset\ \wedge \\ e_1.T^g \cap e_2.T^g = \emptyset\ )$$

We define the following auxiliary function $members_\gamma[\,]$: $G_\gamma \longrightarrow S_\gamma$ such that $members_\gamma[g]$ gives the set of all subjects that are members of the group g. To determine whether a subject s is in

$members_\gamma[g]$, we first determine the unique NGS e, such that $g \in e.G$. Now, $s \in members_\gamma[g]$ if and only if the tag t^g assigned to s (via e.gtag) satisfies the condition that at least one of the two group labels $\langle s, t^g \rangle$ and $\langle *\mathsf{u}, t^g \rangle$ are in the patterns for g, i.e.,

$$\exists\, t^g \in e.T^g \; (\, e.\mathsf{gtag}(s) = t^g \;\wedge \; (\, \langle s, t^g \rangle \in e.\mathsf{patterns}[g] \;\vee\; \langle *\mathsf{u}, t^g \rangle \in e.\mathsf{patterns}[g] \,)\,)$$

As an example, consider an NGS e where

$$\begin{array}{lcl}
e.G & = & \{\, g_{emp},\ g_{mgr},\ g_{exe} \,\} \\
e.T^g & = & \{\, Boss,\ Worker \,\} \\
e.\mathsf{gtag}[s_1] & = & Boss \\
e.\mathsf{gtag}[s_2] & = & Boss \\
e.\mathsf{gtag}[s_3] & = & Worker \\
e.nt^g & = & Worker \\
e.\mathsf{admin} & = & e \\
e.\mathsf{patterns}[g_{exe}] & = & \{\, \langle s_1, Boss \rangle \,\} \\
e.\mathsf{patterns}[g_{mgr}] & = & \{\, \langle *\mathsf{u}, Boss \rangle \,\} \\
e.\mathsf{patterns}[g_{emp}] & = & \{\, \langle *\mathsf{u}, Boss \rangle, \langle *\mathsf{u}, Worker \rangle \,\} \\
e.\mathsf{GRS} & = & \{\, Relabel(Worker, Boss) = g_{mgr} \\
 & & \;\; Relabel(Boss, Worker) = g_{exe} \,\}
\end{array}$$

In this NGS, three groups are defined: executives (g_{exe}), managers (g_{emp}), and employees (g_{mgr}). There are two tags: $Boss$ and $Worker$. There are three subjects; s_1 and s_2 are assigned the tag $Boss$ and s_3 is assigned the tag $Worker$. The new subject tag is $Worker$, so each newly added subject will automatically be assigned the tag $Worker$. The administrative NGS is e itself. According to the patterns, members of the three groups are as follows:

$$\begin{array}{lcl}
members_\gamma[g_{exe}] & = & \{s_1\} \\
members_\gamma[g_{mgr}] & = & \{s_1, s_2\} \\
members_\gamma[g_{mgr}] & = & \{s_1, s_2, s_3\}
\end{array}$$

The group relabeling rules are such that managers can change a subject's tag from $Worker$ to $Boss$ and executives can change a subject's tag from $Boss$ to $Worker$.

State-Change Rules, Ψ There is a single state transition rule ψ in this scheme; ψ consists of six actions that can result in state changes. These actions are mentioned in Section 3.4 of [27] without precise definitions. (We break up the "Relabel an object" operation in [27] into two relabelling actions.) We describe the actions and their effects when applying them to a state $\gamma = \langle S_\gamma, O_\gamma, R_\gamma, G_\gamma, L_\gamma, \mathsf{label}_\gamma, \mathsf{auth}_\gamma, \mathsf{ORS}_\gamma, E_\gamma \rangle$. We use γ' to denote the state after the change.

1. $\mathsf{create\_object}(s, o, \ell = \langle s_1, t_1^o \rangle)$: the subject s creates the object o and assigns the object label ℓ to the object o.

   This action succeeds when $s \in S_\gamma$, $o \notin O_\gamma$, $\ell \in L_\gamma$ and the subject s has the distinguished right wr on the object label ℓ, i.e., $s \in members_\gamma[\mathsf{auth}_\gamma(\ell, \mathsf{wr})]$.

   Effects of the action are $O_{\gamma'} = O_\gamma \cup \{o\}$ and the function label is extended so that $\mathsf{label}_{\gamma'}(o) = \langle s_1, t_1^o \rangle$.

2. $\mathsf{create\_label}(s, \ell = \langle s, t_1 \rangle, g_1, g_2, \cdots, g_k)$, where $k = |R_\gamma|$ is the number of rights in γ: the subject s creates the new object label ℓ, and assigns the groups $g_1, g_2, \cdots, g_k$ to have the rights over ℓ, .

   This action succeeds when $s \in S_\gamma$, $\ell \notin L_\gamma$, the subject in ℓ is s, and $g_1, \cdots, g_k \in G_\gamma$.

   The effects of this action are follows. Let $r_1, r_2, \cdots, r_k$ be the k rights in R_γ. Then $L_{\gamma'} = L_\gamma \cup \{\ell\}$ and the function auth is extended such that $\mathsf{auth}_{\gamma'}(\ell, r_i) = g_i$ for $1 \le i \le k$.

3. $\mathsf{create\_subject}(s, s')$: the subject s creates a new subject s'.

   This action succeeds when $s \in S_\gamma$ and $s' \notin S_\gamma$.

   The effects of this action are $S_{\gamma'} = S_\gamma \cup \{s'\}$ and for every NGS $e \in E_\gamma$, e.gtag is extended so that in γ', $e.\mathsf{gtag}(s') = e.nt^g$.

4. $\mathsf{object\_relabel}(s, \ell_1 = \langle s_1, t_1 \rangle, \ell_2 = \langle s_2, t_2 \rangle)$: the subject s relabels objects having label ℓ_1 to have the label ℓ_2.

   This action succeeds when the first relabelling rule in the object relabelling rule sequence ORS_γ that *matches* (ℓ_1, ℓ_2) is $\mathsf{rl}(p_1, p_2) = h$ and $s \in value[h]$ (the function $value[\,]$ is defined below). The rule $\mathsf{rl}(p_1, p_2) = h$ matches (ℓ_1, ℓ_2) when p_1 matches ℓ_1 and p_2 matches ℓ_2 at the same time. When the pattern $\langle *\mathsf{u}, * \rangle$ matches the label $\langle s_1, t_1 \rangle$, we say that $*\mathsf{u}$ is unified with the subject s_1. Note that when $*\mathsf{u}$ occurs more than one times in p_1, p_2, they should be unified with the same subject.

   Recall that h maybe a group g, a subject s', or one of the four sets: $\{\}, \{*\}, \{*u\}, \{*w\}$. The function $value$ is defined as follows: $value[g] = members_\gamma[g]$; $value[s'] = \{s'\}$; $value[\{\}] = \emptyset$, $value[\{*\}] = S_\gamma$; $value[\{*\mathsf{u}\}]$ is the subject that is unified with $*\mathsf{u}$.

   Consider the following ORS_γ.

$$\begin{array}{lcl}
\mathsf{rl}(\langle *\mathsf{u}, t_1 \rangle,\ \langle s_1, * \rangle) & = & g_1 \\
\mathsf{rl}(\langle s_1, * \rangle,\ \langle *\mathsf{u}, t_2 \rangle) & = & \{*\} \\
\mathsf{rl}(\langle *\mathsf{u}, * \rangle,\ \langle *\mathsf{u}, * \rangle) & = & \{*\mathsf{u}\} \\
\mathsf{rl}(\langle *\mathsf{u}, * \rangle,\ \langle *\mathsf{w}, * \rangle) & = & \{\}
\end{array}$$

   The action $\mathsf{object\_relabel}(s, \langle s_2, t_1 \rangle, \langle s_1, t_2 \rangle)$ would match the first relabelling rule, and it would succeed when s is a member of the group g_1.

The action $\mathsf{object\_relabel}(s, \langle s_1, t_1\rangle, \langle s_2, t_2\rangle)$ would match the second relabelling rule and always succeeds. The action $\mathsf{object\_relabel}(s, \langle s_2, t_2\rangle, \langle s_2, t_1\rangle)$ would match the third relabelling rule and fail, because $*\mathsf{u}$ is unified with s_2. The action $\mathsf{object\_relabel}(s, \langle s_2, t_2\rangle, \langle s_1, t_1\rangle)$ would match the fourth relabelling rule and fail.

The effect of the relabel action is in the function label. For every object o such that $\mathsf{label}_\gamma[o] = \ell_1$, in the new state, $\mathsf{label}_{\gamma'}[o] = \ell_2$.

5. $\mathsf{group\_tag\_relabel}(s, s', t_1^g, t_2^g)$: the subject s relabels the group tag for the subject s' from t_1^g to t_2^g.

   This action succeeds when there is an NGS $e \in E_\gamma$ such that t_1^g and t_2^g are used in e, the subject s' has the group tag t_1^g in e, there is a corresponding group relabelling rule in $e.\mathsf{GRS}$, and s is a member of the group that can use the relabelling rule. More precisely, the action succeeds when

   $$\exists e \in E_\gamma\ (\ e.\mathsf{gtag}[s'] = t_1^g\ \wedge\ \text{“}Relabel(t_1^g, t_2^g) = g\text{”} \in e.\mathsf{GRS}\ \wedge\ s \in members_\gamma[g]\ )$$

   Note that the tags t_1^g and t_2^g can appear only in one NGS and they must appear in the same NGS for the action to succeed. The effect of this action is such that the function $e.\mathsf{gtag}$ is changed such that in γ', $e.\mathsf{gtag}[s'] = t_2^g$.

6. $\mathsf{create\_ngs}(s, e)$: the subject s creates a new NGS e.

   To perform this action, one must provide the complete description of a new NGS e, i.e., the 7-tuple $\langle e.G, e.T^g, e.\mathsf{gtag}, e.nt^g, e.\mathsf{admin}, e.\mathsf{patterns}, e.\mathsf{GRS}\rangle$. For this action to succeed, the groups defined in e and the group tags in e must be new, i.e., they do not appear in any existing NGS's in γ.

   The effects are that $G_{\gamma'} = G_\gamma \cup e.G$ and $E_{\gamma'} = E_\gamma \cup e$.

Given the above state transition rule, we make the following observations. No removal of subjects, objects, labels, or groups is defined. Given a state $\langle S_\gamma, O_\gamma, R_\gamma, G_\gamma, L_\gamma, \mathsf{label}_\gamma, \mathsf{auth}_\gamma, \mathsf{ORS}_\gamma, E_\gamma\rangle$, S_γ (the set of subjects), O_γ (the set of objects), and G_γ (the set of groups) may change as a result of create_subject, create_object, and create_label, respectively. R_γ, the set of rights, is fixed for the system and does not change. G_γ, the set of groups, may change when a new NGS is added by the create_ngs action. The function label_γ: $O_\gamma \longrightarrow L_\gamma$ is extended when a new object is added and is changed when an object relabelling action object_relabel happens. The function auth_γ is extended when a new object label is created; existing assignments do not change. ORS_γ, the object relabelling rule sequence, always stay the same. E_γ is extended when a new NGS is added.

5.2. Encoding a simple DAC scheme in the Solworth-Sloan scheme

In this section, we encode a relatively simple DAC scheme in the Solworth-Sloan scheme. The DAC scheme we consider is a sub-scheme of the Graham-Denning scheme. It is called Strict DAC with Change of Ownership (SDCO) and is one of the DAC schemes discussed by Osborn et al. [21]. Our construction is based on comments by Solworth and Sloan [27] on how various DAC schemes can be encoded in the Solworth-Sloan scheme. As the presentation in that paper is not detailed, we offer a more detailed construction. Our construction lets us assess the utility of the Solworth-Sloan scheme in encoding SDCO. After we present the encoding, we discuss its deficiencies from the standpoints of correctness, and the overhead it introduces.

Strict DAC with Change of Ownership (SDCO) As we mention above, SDCO is a sub-scheme of the Graham-Denning scheme (see Section 4.1). In SDCO, there is a distinguished right, *own*, but no *control* right. Also, there are no rights with the copy flag. The state-change rules in SDCO are the commands grant_r (for each $r \in R_\psi$), delete_r (for each $r \in R_\psi$), grant_own, create_object and create_subject. We do not consider commands to destroy subjects or objects as their counterparts are not specified for the Solworth-Sloan scheme.

For simplicity, we consider an SDCO scheme that has only three rights $\mathsf{own}, \mathsf{re}, \mathsf{wr}$. In the Solworth-Sloan scheme, if two objects o_1 and o_2 have the same label, then o_1 and o_2 always have the same access characteristics. That is, in every state, the set of subjects having a right r over o_1 is the same as the set of subjects having the right r over o_2. In SDCO, one can reach states in which o_1 and o_2 have different access characteristics. Therefore, each object needs to be assigned a distinct label.

Therefore, before creating an object, one has to create a new label. When creating a new label ℓ, one has to assign a group to $\mathsf{auth}(\ell, \mathsf{own})$ and a group to $\mathsf{auth}(\ell, \mathsf{re})$; and a group to $\mathsf{auth}(\ell, \mathsf{wr})$. Each pair $\langle \ell, r\rangle$ determines a unique access class. Therefore, a distinct group needs to be created. We use $g(o, r)$ to denote the group that will be assigned to have the right r over object o.

In order to keep track of which subset of rights a subject has over an object, we need 8 group tags, one corresponding to each subset of $\{own, \mathsf{re}, \mathsf{wr}\}$, we use $t^g(o, x)$, where x is a 3-bit string to denote these tags.

In order for a subject s to create an object o, s needs to do the following:

1. Create an NGS $e = \langle e.G, e.T^g, e.\mathsf{gtag}, e.nt^g, e.\mathsf{admin}, e.\mathsf{patterns}, e.\mathsf{GRS}\rangle$ as follows.

   - $e.G = \{g(o, \mathsf{own}), g(o, \mathsf{re}), g(o, \mathsf{wr})\}$

- $e.T^g = \{t^g(o,000), t^g(o,001), t^g(o,010), t^g(o,011), t^g(o,100), t^g(o,101), t^g(o,110), t^g(o,111)\}$.
- $e.\mathsf{gtag}[s] = t^g(o,100)$ and $e.\mathsf{gtag}[s'] = t^g(o,000)$ for every $s' \in S_\gamma$ s.t. $s' \neq s$.
- $e.nt^g = t^g(o,000)$
- $e.\mathsf{admin} = e$
- $e.\mathsf{patterns}[g(o,\mathsf{own})] = \{\langle *u, t^g(o,100)\rangle, \langle *u, t^g(o,101)\rangle, \langle *u, t^g(o,110)\rangle, \langle *u, t^g(o,111)\rangle\}$
  $e.\mathsf{patterns}[g(o,\mathsf{re})] = \{\langle *u, t^g(o,010)\rangle, \langle *u, t^g(o,011)\rangle, \langle *u, t^g(o,110)\rangle, \langle *u, t^g(o,111)\rangle\}$
  $e.\mathsf{patterns}[g(o,\mathsf{wr})] = \{\langle *u, t^g(o,001)\rangle, \langle *u, t^g(o,011)\rangle, \langle *u, t^g(o,101)\rangle, \langle *u, t^g(o,111)\rangle\}$
  That is, in each tag, the first bit corresponds to own, the second to re, and the third to wr. In the set of patterns for the group that corresponds to own, the first bit is always set in each tag, and similarly for the groups that correspond to re and wr respectively.
- $e.\mathsf{GRS} = \{\mathit{Relabel}(g(o,b_1b_2b_3), g(o,b'_1b'_2b'_3)) = g(o,\mathsf{own}) \mid b_1b_2b_3, b'_1b'_2b'_3 \in \{0,1\}^3 \wedge b_1b_2b_3$ and $b'_1b'_2b'_3$ differ in exactly one bit $\}$

2. Use $\mathsf{create\_label}(s, \langle s, t(o)\rangle, g(o,\mathsf{re}), g(o,\mathsf{wr}))$ to create the label $\ell(o)$.
3. Use the action $\mathsf{create\_object}(s, o, \langle s, t(o)\rangle)$ to create the object o and label it with $\ell(o)$.

To grant or revoke a right, one uses group relabelling. For instance, suppose s is a subject, and for the NGS, e, $e.\mathsf{gtag}[s] = t^g(o,000)$. Then, we know that s is not a member of any of the groups $g(o,\mathsf{own})$, $g(o,\mathsf{re})$ or $g(o,\mathsf{wr})$. The subject would be granted the right re by relabelling $\langle s, t^g(o,000)\rangle$ to the label $\langle s, t^g(o,010)\rangle$. The execution of this relabelling results in the subject becoming a member of the group $g(o,\mathsf{re})$, thereby giving him the right re over the object o. Similarly, the subject would have the right re revoked by relabelling $\langle s, t^g(o,010)\rangle$ to the label $\langle s, t^g(o,000)\rangle$. These operations can be carried out only by a subject that is a member of the group $g(o,\mathsf{own})$.

We make the following observations about the above mapping.

- The above mapping does not capture the state invariant in SDCO that in every state, there is exactly one owner for every object that exists. In the Solworth-Sloan system that results from the above mapping, one can perform relabelling operations and reach states in which there are mutiple owners for an object, or no owner for an object. For instance, suppose that there already exists a subject s such that $s \in \mathit{members}_\gamma[g(o,\mathsf{own})]$. Given the above relabelling rules, there is nothing that precludes another subject from also becoming a member of the group $g(o,\mathsf{own})$ while s continues to maintain membership in that group. It is also possible to remove the membership of s in the group $g(o,\mathsf{own})$ thereby leaving the object with no owner. It is unclear how we would prevent such situations from occuring in a system based on the Solworth-Sloan scheme.

- We are unable to capture destruction of subjects and objects as such constructs have not been specified for the Solworth-Sloan scheme. Destruction of subjects and objects is generally considered to be an important component of any access control system. We point out that a state-change rule to destroy a subject or an object in the Solworth-Sloan scheme must be carefully designed, as there are several components of the state (such as tags) of which we must keep track. Therefore, adding such state-change specifications does not appear to be a trivial task. In particular, it is unclear how and with what overhead we can capture in the Solworth-Sloan scheme, the notion of transfer of ownership over objects owned by a subject that is being destroyed.

- There is considerable overhead in implementing a relatively simple DAC scheme (SDCO) in the Solworth-Sloan scheme. For each object, we need to create a set of labels whose size is linear in the number of the subjects in the state. We also need to create a set of tags whose size is exponential in the number rights in the system. These tags are used to define groups, and the therefore, the number of entries in all the sets of patterns is also exponential in the number of rights in the system. This is considerable overhead considering the simplicity of SDCO, and the fact that one can "directly" implement it, with efficiently decidable safety.

Our conclusion is that several of the claims made by Solworth and Sloan [27] are incorrect. In particular, not only is the motivation (decidable safety) for the creation of the new scheme invalid, but it is also not effective in implementing relatively simple DAC schemes.

6. Conclusions

The focus of this paper is to provide a clear picture of safety analysis in DAC. We have used a state-transition-system-based meta-formalism to precisely model access control schemes and systems and have studied safety analysis in a general DAC scheme from the literature, the Graham-Denning scheme [8]. We have presented an algorithm for deciding safety with running time $O(n^3)$ in the Graham-Denning scheme, and proved that the algorithm is correct. We have also countered several claims made by Solworth and Sloan [27]. In particular, we have countered the claim that the mapping presented there encodes all DAC schemes by considering a relatively simple DAC scheme and demonstrating that the mapping has several deficiencies. We conclude by asserting that safety in existing general DAC schemes is decidable and there is no need to invent new DAC schemes with decidable safety as the primary goal.

Acknowlegements Portions of this work were supported by NSF Grants CCR-0325951 and IIS-0430274 and by sponsors of CERIAS.

References

[1] P. Ammann and R. S. Sandhu. Safety analysis for the extended schematic protection model. In *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, pages 87–97, May 1991.

[2] P. Ammann and R. S. Sandhu. The extended schematic protection model. *Journal of Computer Security*, 1(3-4):335–383, 1992.

[3] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati. An access control model supporting periodicity constraints and temporal reasoning. *ACM Transactions on Database Systems*, 23(3):231–285, 1998.

[4] E. Bertino, C. Bettini, and P. Samarati. A temporal authorization model. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 126–135. ACM Press, 1994.

[5] T. Budd. Safety in grammatical protection systems. *International Journal of Computer and Information Sciences*, 12(6):413–430, 1983.

[6] N. C. S. Center. A guide to understanding discretionary access control in trusted systems, Sept. 1987. NCSC-TG-003.

[7] D. D. Downs, J. R. Rub, K. C. Kung, and C. S. Jordan. Issues in discretionary access control. In *Proceedings of IEEE Symposium on Research in Security and Privacy*, pages 208–218, Apr. 1985.

[8] G. S. Graham and P. J. Denning. Protection — principles and practice. In *Proceedings of the AFIPS Spring Joint Computer Conference*, volume 40, pages 417–429. AFIPS Press, May 16–18 1972.

[9] P. P. Griffiths and B. W. Wade. An authorization mechanism for a relational database system. *ACM Transactions on Database Systems*, 1(3):242–255, 1976.

[10] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, Aug. 1976.

[11] A. K. Jones, R. J. Lipton, and L. Snyder. A linear time algorithm for deciding security. In *17th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 33–41, October 1976.

[12] L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(3):872–923, May 1994.

[13] B. W. Lampson. Protection. In *Proceedings of the 5th Princeton Conference on Information Sciences and Systems*, 1971. Reprinted in ACM Operating Systems Review, 8(1):18-24, Jan 1974.

[14] N. Li and M. V. Tripunitara. Security analysis in role-based access control. In *Proceedings of the Ninth ACM Symposium on Access Control Models and Technologies (SACMAT 2004)*, pages 126–135, June 2004.

[15] N. Li and M. V. Tripunitara. On safety in discretionary access control. Technical Report CERIAS-TR-2005-20, Center for Education and Research in Information Assurance and Security, Purdue University, Feb. 2005.

[16] N. Li, W. H. Winsborough, and J. C. Mitchell. Beyond proof-of-compliance: Safety and availability analysis in trust management. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 123–139. IEEE Computer Society Press, May 2003.

[17] R. J. Lipton and L. Snyder. A linear time algorithm for deciding subject security. *Journal of the ACM*, 24(3):455–464, 1977.

[18] T. Lunt. Access control policies: Some unanswered questions. In *Proceedings of the 2nd IEEE Computer Security Foundations Workshop*, pages 227–245. IEEE Computer Society Press, June 1988.

[19] N. H. Minsky. Selective and locally controlled transport of privileges. *ACM Transactions on Programming Languages and Systems*, 6(4):573–602, Oct. 1984.

[20] R. Motwani, R. Panigrahy, V. A. Saraswat, and S. Ventkatasubramanian. On the decidability of accessibility problems (extended abstract). In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 306–315. ACM Press, May 2000.

[21] S. Osborn, R. S. Sandhu, and Q. Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and System Security*, 3(2):85–106, May 2000.

[22] P. Samarati and S. de Capitani di Vimercati. Access control: Policies, models, and mechanisms. In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, volume 2171 of *Lecture Notes in Computer Science*, pages 137–196. Springer, 2001.

[23] R. S. Sandhu. The schematic protection model: Its definition and analysis for acyclic attenuating systems. *Journal of the ACM*, 35(2):404–432, 1988.

[24] R. S. Sandhu. Expressive power of the schematic protection model. *Journal of Computer Security*, 1(1):59–98, 1992.

[25] R. S. Sandhu. The typed access matrix model. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, pages 122–136. IEEE Computer Society Press, May 1992.

[26] R. S. Sandhu. Undecidability of the safety problem for the schematic protection model with cyclic creates. *Journal of Computer and System Sciences*, 44(1):141–159, Feb. 1992.

[27] J. A. Solworth and R. H. Sloan. A layered design of discretionary access controls with decidable safety properties. In *Proceedings of IEEE Symposium on Research in Security and Privacy*, May 2004.

[28] J. A. Solworth and R. H. Sloan. Security property based administrative controls. In *Proceedings of the Ninth European Symposium on Research in Computer Security (ESORICS 2004)*, pages 244–259. Springer, Sept. 2004.

[29] M. Soshi. Safety analysis of the dynamic-typed access matrix model. In *Proceedings of the Sixth European Symposium on Research in Computer Security (ESORICS 2000)*, pages 106–121. Springer, Oct. 2000.

[30] M. Soshi, M. Maekawa, and E. Okamoto. The dynamic-typed access matrix model and decidability of the safety problem. *IEICE Transactions on Fundamentals*, E87-A(1):190–203, Jan. 2004.